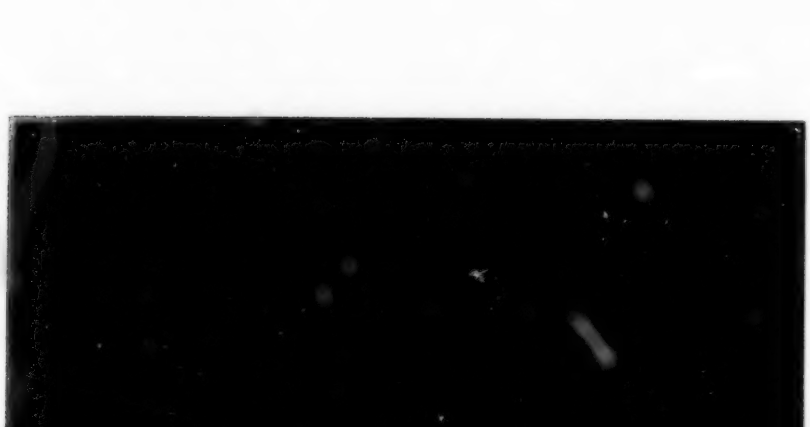
CENTRE DE RECHERCHE
SUR LES TRANSPORTS
(C.R.T.)
CENTRE FOR RESEARCH
ON TRANSPORTATION

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE



TRAVAIL D'ÉTUDIANT

CENTRE DE RECHERCHE SUR LES TRANSPORTS
UNIVERSITÉ DE MONTRÉAL
C.P. 6128, SUCCURSALE A,
MONTRÉAL
CANADA H3C 3J7

TÉLÉPHONE: (514) 343-7575
TÉLEX: (21) 05-561325
EMAIL: crt@crt.umontreal.ca
TÉLÉCOPIEUR (FAX): (514) 343-7121

92-07817/

CRT-848

Les algorithmes parallèles asynchrones en recherche opérationnelle*

par

Ismaïl Chabini **

* Travail présenté dans le cadre des examens postdoctoraux.

** Département d'informatique et de recherche opérationnelle et Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale A, Montréal, Canada H3C 3J7.

Centre de recherche sur les transports — Publication #848
Département d'informatique et de recherche opérationnelle — Publications #834

Août 1992

Dépôt légal – Bibliothèque nationale du Québec,
Bibliothèque nationale du Canada, 1992

Résumé

Nous présentons une revue et une synthèse de la littérature qui traite des algorithmes parallèles asynchrones en recherche opérationnelle. La plupart des recherches faites sont consacrées aux algorithmes itératifs de la forme $x := f(x)$. Nous analysons d'abord des implantations parallèles synchrones où nous comparons les algorithmes de types Jacobi et Gauss-Seidel. Ensuite, nous définissons les algorithmes parallèles asynchrones et nous présentons leurs avantages potentiels. Ces algorithmes sont divisés en trois classes: totalement, partiellement et faiblement asynchrones. Cette distinction est introduite pour assurer la convergence de certaines méthodes, convergence qui peut dépendre du degré d'asynchronisme permis. Nous présentons, pour chaque classe, une synthèse des principaux résultats de convergence et nous les illustrons par des méthodes connues en recherche opérationnelle. La terminaison des algorithmes asynchrones est aussi abordée. De plus, nous identifions plusieurs avenues de recherche.

Mots-clés: Algorithmes asynchrones, algorithmes parallèles, algorithmes distribués, méthodes itératives, point fixe, recherche opérationnelle, terminaison.

1. Introduction

Dans ce travail, nous présentons une revue et une synthèse de la littérature traitant de la conception et de l'analyse des algorithmes asynchrones ainsi que de leurs implantations sur des ordinateurs parallèles. Nous nous intéressons particulièrement à la résolution des problèmes qui émanent du domaine de la recherche opérationnelle.

La plupart des travaux faits dans ce domaine ont porté sur la classe des méthodes itératives de la forme $x := f(x)$, où $x = (x_1, ..., x_n)$ est un vecteur de $\Re^n$ et f est une fonction définissant le processus itératif. Beaucoup de méthodes classiques de résolution de problèmes de recherche opérationnelle s'inscrivent dans ce cadre. Il s'agit donc d'une classe importante, d'autant plus qu'elle permet d'illustrer les principes sur lesquels reposent les méthodes asynchrones.

Nous nous intéressons aux problèmes de grandes tailles dont la résolution sur des machines séquentielles nécessite un temps de calcul élevé, d'où le recours au traitement parallèle. Danc ce cas, plusieurs processeurs effectuent des calculs en simultanéïté et sont souvent appelés à coopérer en échangeant de l'information nécessaire.

Une famille importante d'ordinateurs parallèles sont les systèmes multiprocesseurs (Chabini, 1990). Ces derniers peuvent avoir un mode de fonctionnement synchrone ou asynchrone. Dans le premier mode, les processeurs ont une horloge

globale commune qui les contrôle pour exécuter la même opération. Les processeurs exécutent la même opération mais sur des données différentes (SIMD : Single Instruction Multiple Data). Par contre, dans le mode asynchrone, il n'y a pas d'horloge globale. Les processeurs ont chacun une horloge locale et peuvent évoluer à des vitesses de traitement différentes. À un moment donné, plusieurs instructions sont exécutées sur des données différentes (MIMD: Multiple Instruction Multiple Data).

La manière dont la communication est faite dans un système multiprocesseurs permet d'en distinguer deux types: les systèmes à échange de messages, où la communication s'effectue à travers des liens reliant les processeurs, et les systèmes à mémoire partagée, où la communication se fait à travers une mémoire commune. Cette distinction est particulièrement importante dans l'analyse des algorithmes asynchrones. En effet, certaines hypothèses parfois nécessaires à la validité de propriétés importantes de ces méthodes, ne sont pas vérifiées dans le cas des systèmes à mémoire partagée. Dans ce travail, nous supposons l'utilisation d'un système à échange de messages et lorsque certains résultats ne sont pas valables pour un système à mémoire partagée, nous le précisons.

Notons que la communication est une composante importante dans l'analyse d'un algorithme parallèle (synchrone ou asynchrone). Lint et Agerwala (1982) font une étude exhaustive de cet aspect.

Pour exploiter un système multiprocesseurs asynchrone, deux possibilités s'offrent. La première est de simuler une machine synchrone en greffant à l'algorithme à implanter des procédures de synchronisation (Awerbuch,1985). Ces dernières sont appelées là où c'est nécessaire. Une seconde possibilité est de ne mettre aucun point de synchronisation dans l'algorithme à implanter. Ainsi, l'asynchronisme de la machine va apparaître dans la méthode implantée. Un tel algorithme va tolérer une hétérogénéïté aussi bien dans la rapidité des processeurs que dans les délais de communication, sans avoir à recourir à un protocole de synchronisation. Toutefois, à cause de son caractère chaotique, un algorithme asynchrone va nécessiter une preuve de validité et de convergence, qui est souvent plus difficile à établir que dans le cas d'un algorithme synchrone.

Ce rapport est organisé comme suit. Dans la section 2, il est question des algorithmes parallèles synchrones et particulièrement des méthodes de types Jacobi et Gauss-Seidel. Ces dernières montrent deux effets antagonistes: potentiel de parallélisme et accérélation de la convergence. Les algorithmes asynchrones allient les deux effets. Dans la même section, nous décrivons brièvement deux principales méthodes de synchronisation. Dans la section 3, nous présentons d'abord les avantages potentiels des implantations asynchrones. Celles-ci sont introduites pour pallier à certains défauts inhérents aux méthodes synchrones. Ensuite, nous décrivons un modèle d'algorithmes itératifs asynchrones. Dans ce modèle, nous distinguons trois classes d'algorithmes: totalement, partielle-

ment et faiblement asynchrones. Cette distinction est introduite pour assurer la convergence de certaines méthodes, convergence qui peut dépendre du degré d'asynchronisme permis. Les sections 4, 5, 6 sont consacrées respectivement aux trois classes énumérées ci-dessus. Dans chaque section, nous présentons une revue de la littérature, incluant des théorèmes de convergence, et nous illustrons chaque classe d'algorithmes à l'aide de méthodes de recherche opérationnelle. Outre les difficultés reliées à la convergence, les algorithmes itératifs asynchrones sont difficiles à arrêter en temps fini. Le problème de la terminaison est abordé à la section 7. En conclusion, nous résumons les principaux résultats présentés dans ce travail et nous identifions plusieurs avenues de recherche.

2. Algorithmes itératifs synchrones

Dénotons par $X_1, ..., X_p$ des sous-ensembles des espaces $\Re^{n_1}, ..., \Re^{n_p}$ respectivement. Dénotons également $n = \sum_{i=1}^{p} n_i$ et $X = \prod_{i=1}^{p} X_i$ (le produit cartésien des X_i). Tout $x \in \Re^n$ est alors décomposé sous la forme $x = (x_1, ..., x_p)$, où $x_i \in \Re^{n_i}$. Pour $i = 1, ..., p$, f_i est une fonction de X vers X_i et f est la fonction de X vers X définie par : $f(x) = (f_1(x), ..., f_p(x))$. Nous nous intéressons au problème du point fixe $x^* = f(x^*)$. Pour le résoudre, nous utilisons le processus itératif suivant, où t représente l'indice des itérations et $U(t)$ est l'ensemble des variables mises à jour à l'itération t, $U(t) \subset \{1, \ldots, p\}$:

$$x_i(t+1) := \begin{cases} f_i(x(t)) & si\ i \in U(t) \\ x_i(t) & sinon \end{cases} \qquad (2.1)$$

avec

$$x(0) \in X$$

Une itération de type Jacobi correspond à $U(t) = \{1, ..., p\}$, pout tout t . Une itération de Gauss-Seidel peut être réalisée avec $U(t)$ contenant un seul élément de $\{1, \ldots, p\}$ et tel que $\bigcup_{t=0}^{p-1} U(t_0 + t) = \{1, ..., p\}$, pour tout t_0 .

2.1 Implantation parallèle

Supposons que nous avons une machine parallèle avec p processeurs. Pour implanter l'algorithme (2.1) sur cette machine, on affecte au processeur i la mise à jour de la composante x_i ($x_i := f_i(x)$). Si la fonction f_i dépend de la composante x_j, alors le processeur i doit être informé de la valeur de x_j par le processeur j . Cette relation de dépendance entre les variables peut être représentée par un graphe $G = (N, A)$, appelé graphe de dépendance (Bertsekas et Tsitsiklis, 1989a), où $N = \{1, ..., p\}$ et $A = \{(i, j) \mid i \neq j$ et f_j dépend de $x_i\}$.

À une étape t, tous les processeurs dont le numéro appartient à $U(t)$ exécutent en parallèle une mise à jour et envoient les résultats à ceux qui en ont besoin. Ce n'est qu'après que ces derniers aient reçu ces résultats, qu'on passe à l'étape $(t + 1)$. Ceci nécessite une synchronisation totale des processeurs, dont la mise en oeuvre sera discutée dans la section 2.3.

Avec une itération de Jacobi, le maximum de parallélisme est atteint, car toutes les variables peuvent être mises à jour en même temps . Par contre, si le graphe

de dépendance de f est complet (chaque composante x_i dépend de toutes les autres), l'itération de Gauss-Seidel ne peut pas être réalisée avec $U(t)$ contenant plus qu'un élément. Par conséquent, elle ne pourra pas être effectuée en parallèle.

Si le graphe de dépendance correspondant à f est quelconque, trouver une façon de réaliser le maximum de parallélisme est équivalent à un problème de coloration du graphe de dépendance (Bertsekas et Tsitsiklis 1989a). Malheureusement, ce dernier problème est difficile en lui-même.

Sur un ordinateur séquentiel, la vitesse de convergence des itérations de type Gauss-Seidel est toujours supérieure à celle des itérations de type Jacobi. Ce résultat est connu sous le nom du théorème de Stein-Rosenberg (Bertsekas et Tsitsiklis, 1989a). Une surprise survient en calcul parallèle: ce sont les itérations de type Jacobi qui maximisent la vitesse de convergence, pourvu que f soit monotone et qu'une itération de Gauss-Seidel prenne au moins autant de temps que p itérations de type Jacobi. Ce résultat est démontré par Tsitsiklis (1989) et généralise la même propriété établie dans le cas linéaire par Smart et White (1988).

2.2 Modèle mathématique

Dénotons par T^i l'ensemble des instants où le processeur i a effectué une mise à jour. T^i est un sous ensemble de $\{0, 1, \ldots\}$. L'implantation parallèle synchrone du processus (2.1) est équivalente mathématiquement à:

$$x_i(t+1) = \begin{cases} f_i(x_1(t), x_2(t), \ldots, x_p(t)) & si\ t \in T^i \\ x_i(t) & sinon \end{cases} \tag{2.2}$$

En faisant varier T^i, on obtient une classe de méthodes itératives connue sous le nom de "conduite libre" (free steering) (Miellou, 1975b). Afin d'assurer la convergence, ces méthodes doivent être telles que chaque composante est mise à jour un nombre infini de fois. Avec des choix appropriés des T^i , les itérations de type Jacobi et Gauss-Seidel peuvent être considérées comme des cas particuliers de cette classe.

2.3 Méthodes de synchronisation

Supposons un système multiprocesseurs fonctionnant en mode asynchrone. La réalisation de l'implantation synchrone est possible en introduisant des protocoles de synchronisation (Awerbuch, 1985). Nous décrivons ici deux méthodes de synchronisation en les illustrant pour le cas du processus itératif de type Jacobi (Bertsekas et Tsitsiklis, 1989a).

1. **Synchronisation globale**. Les processeurs ne commencent l'itération $(t+1)$, appelée phase, que lorsque chaque processeur i a complété l'itération t et a reçu la valeur $x_j(t)$ de chaque processeur j tel que x_j apparaît dans l'expression de f_i. Elle peut être réalisée de la manière suivante: les processeurs forment un arbre binaire, dont la racine est appelée le maître. À la phase t, chaque processeur i évalue $x_i(t)$ et reçoit les valeurs $x_j(t)$ des processeurs j tels que f_i dépend de x_j. Ensuite, chaque processeur feuille

envoie un message de fin de phase à son père. Lorsqu'un processeur situé à un noeud interne reçoit les messages de fin de phase de ses enfants, il le transmet à son père. Lorsque le processeur maître reçoit ce message, il envoit un message de début de la phase suivante à ses enfants. Ces derniers propagent à leur tour le message et entament l'itération suivante.

2. **Synchronisation locale**. Le processeur i entame l'itération $(t+1)$ juste après qu'il ait reçu des autres processeurs, les valeurs $x_j(t)$ dont il a besoin.

Une étude comparative des deux méthodes montre la supériorité de la deuxième (Bertsekas et Tsitsiklis, 1989a). Dubois et Briggs (1982) présentent une analyse théorique et expérimentale sur le même sujet.

Qu'on utilise l'une ou l'autre méthode pour synchroniser un algorithme, les protocoles de synchronisation peuvent devenir coûteux lorsque les points de synchronisation sont nombreux. Les algorithmes asynchrones visent à éliminer ces derniers.

3. Algorithmes asynchrones

Les algorithmes asynchrones ont été introduits pour la première fois par Chazan et Miranker (1969), sous le nom de relaxation chaotique, pour la solution des équations linéaires. Dans une implantation parallèle asynchrone du processus itératif $x := f(x)$, les processeurs ne sont pas obligés d'attendre la réception de tous les messages dont ils ont besoin pour effectuer l'itération courante. En

effet, lors de la mise à jour de la variable x_i, le processeur i vérifie d'abord si la valeur courante de la variable x_j est disponible. Sinon, il utilise la dernière valeur de x_j reçue.

Les algorithmes asynchrones permettent donc de considérer des processeurs opérant à des vitesses de traitement différentes et de tolérer des délais de communications arbitraires. De plus, l'envoi des messages peut ne pas se faire au terme de chaque itération et l'ordre de leur réception peut différer de celui de leur envoi.

Contrairement aux algorithmes synchrones, les méthodes asynchrones ne peuvent être caractérisées par un modèle mathématique descriptif (voir section 2.2). En effet, lors de la mise à jour à l'itération t, un processeur i utilise comme valeur de x_j, $x_j\Big(t - \tau_j^i(t)\Big)$, où $\tau_j^i(t)$ est un délai imprévisible. Ceci explique les difficultés rencontrées lors de leur analyse de convergence. Une autre difficulté rattachée aux algorithmes asynchrones est d'établir des conditions d'arrêt et de les détecter globalement.

Dans cette section, nous présentons d'abord certains avantages potentiels des algorithmes asynchrones. Ensuite, nous décrivons un modèle algorithmique qui servira lors de l'analyse de convergence.

3.1 Avantages des algorithmes asynchrones

Les algorithmes asynchrones ont été introduits pour pallier aux inconvénients des algorithmes synchrones. Kung (1976), Bertsekas et Tsitsiklis (1989a) et

Bertsekas (1982, 1983) présentent certains avantages possibles des algorithmes asynchrones. Nous les résumons ci-dessous:

– Réduction de la pénalité de synchronisation

Dans le cas d'une synchronisation globale, le temps pris par une itération est celui du processeur qui termine le dernier. Ainsi, celui qui finit le premier doit attendre le dernier. Ceci représente un manque à gagner dans le temps d'exécution global. La fluctuation du temps de traitement d'un processeur à l'autre est pratiquement inévitable. En effet, même si les processeurs avaient la même rapidité et que leur charge était la même, le temps de traitement d'un processeur peut dépendre des données et n'est qu'une variable aléatoire qui suit une certaine loi de probabilité. En adoptant un style asynchrone, qui n'oblige pas les processeurs à attendre le dernier, le temps global peut diminuer.

Même avec une méthode de synchronisation locale, l'asynchronisme peut être profitable. En effet, un processeur i devant attendre la réception de la valeur d'une variable x_j, il se peut que celle-ci n'ait pas (ou à peu près pas) changé par rapport à l'itération passée.

– Combinaison des avantages des méthodes de Gauss-Seidel et de Jacobi

Les algorithmes asynchrones permettent le maximum de parallélisme, car toutes les composantes peuvent être mises à jour en même temps, comme dans la méthode de Jacobi. De plus, aussitôt qu'une nouvelle valeur d'une variable x_j est

disponible à un processeur donné, ce dernier l'incorpore dans le processus itératif. Ceci va accélérer la convergence comme dans la méthode de Gauss-Seidel.

3.2 Modèle algorithmique

Nous allons décrire un modèle pour les algorithmes asynchrones présenté dans Bertsekas (1983) et Bertsekas et Tsitsiklis (1989a, 1989b). X et f sont tels que définis dans la section 2. Soit $T = \{0, 1, 2, ...\}$ l'ensemble des instants où une ou plusieurs variables x_i sont mises à jours par les processeurs du système parallèle. Les instants $t \in T$ n'ont pas à être connus par les processeurs. Désignons par T^i l'ensemble des instants où x_i est mise à jour. Pour un instant $t \in T$, $x_i(t)$ désigne la valeur de x_i qui réside au processeur i. Lors d'une mise à jour, un processeur i peut ne pas avoir accès aux plus récentes valeurs des composantes de x (à cause de l'asynchronisme). Ainsi,

$$x_i(t+1) = f_i\left(x_1(\tau_1^i(t)), ..., x_p(\tau_p^i(t))\right) \quad , \ \forall \ t \in T^i, \tag{3.1}$$

où $\tau_j^i(t)$ sont des entiers satisfaisant $0 \leq \tau_j^i(t) \leq t$, pour tout t. Si $t \notin T^i$, $x_i(t)$ ne change pas, ainsi

$$x_i(t+1) = x_i(t), \qquad \forall \ t \notin T^i \tag{3.2}$$

On suppose que l'algorithme est initialisé avec $x(0) \in X$.

La différence $t - \tau_j^i(t)$, appelée degré d'asynchronisme, peut être vue comme un genre de délai de communication du message x_j, envoyé par le processeur j

au processeur i, qui est utilisé par ce dernier dans la mise à jour de la composante x_i à l'instant t. Dans un algorithme synchrone $t - \tau_j^i(t) = 0$. Lorsque $t - \tau_j^i(t)$ augmente, on dit que l'asynchronisme dans l'algorithme est plus grand.

Pour que l'algorithme progresse, il ne faut pas que $\tau_j^i(t)$ soit borné supérieurement. Ce qui veut dire par exemple, qu'un processeur peut arrêter d'itérer en autant qu'il reprenne à un moment donné. Ceci justifie l'importance de l'hypothèse suivante:

Hypothèse 3.1 **(Asynchronisme total):** *Les ensembles T^i sont infinis et si $\{t_k\}$ est une suite d'éléments de T^i qui tend vers l'infini, alors $\lim_{k\to\infty} \tau_j^i(t_k) = \infty$, pour tout j.*

Remarquons que l'hypothèse 1, n'impose aucune borne supérieure sur le degré d'asynchronisme $t - \tau_j^i(t)$.

Les algorithmes asynchrones qui n'ont besoin que de cette hypothèse pour converger, outre celles exigées par la version synchrone du même algorithme, sont dits algorithmes totalement asynchrones.

Malheureusement, l'hypothèse d'asynchronisme total n'est pas toujours suffisante pour assurer la convergence. C'est pourquoi on doit parfois imposer une borne supérieure sur le degré d'asynchronisme.

Hypothèse 3.2 **(Asynchronisme partiel):** *Il existe une constante positive B telle que:*

(a) *Pour tout* i *et pour tout* $t \geq 0$, *au moins un des éléments de* $\{t, t+1, ..., t+B-1\}$ *appartient à* T^i.

(b) $t - B < \tau_j^i(t) \leq t$, $\forall\, i, j, \forall t \in T^i$.

(c) $\tau_i^i(t) = t$, $\forall i, \forall t \in T^i$.

La constante B est appelée mesure d'asynchronisme. La condition (c) exige que la valeur de la composante x_i au processeur i soit toujours la plus récente. Cette hypothèse est vérifiée dans le cas d'un système à échange de messages mais si le système est à mémoire partagée, et que plusieurs processeurs mettent à jour la même composante x_i de x, elle n'est plus valide. Notons qu'elle est nécessaire à la convergence de certains algorithmes partiellement asynchrones (Bertsekas et Tsitsiklis, 1989a; Lubachevsky et Mitra, 1986).

Les algorithmes qui sont étudiés dans la suite, se divisent en trois catégories:

1. ceux qui convergent sans aucune borne sur le degré d'asynchronisme (totalement asynchrones);
2. ceux qui divergent en asynchronisme total mais qui convergent dès qu'on introduit une mesure d'asynchronisme B finie mais arbitraire (partiellement asynchrones);
3. ceux qui ne convergent que si la mesure d'asynchronisme est faible (faiblement asynchrones).

4. Algorithmes totalement asynchrones

La convergence des algorithmes totalement asynchrones (vérifiant l'hypothèse 3.1) a été démontrée par plusieurs auteurs en considérant différentes classes de fonctions définissant le processus itératif:

- Le cas linéaire

  C'est dans ce cas que le premier travail a été fait (Chazan et Miranker, 1969). Donnelly (1971) s'est interessé à la sur/sous relaxation du processus itératif.

- Le cas non-linéaire

  Nous retrouvons les travaux de Miellou (1975b), Baudet (1978), El Tarazi (1982) et Miellou et Spiteri (1985).

- Les fonctions monotones

  Miellou (1975a) et Bertsekas (1982) ont étudié cette classe.

- Le cas discret

  Uresin et Dubois (1986, 1990) et Robert (1987) ont traité le cas où le domaine de définition du processus itératif est discret.

- Itérations générales

  Bertsekas (1983) donne un théorème qui fournit un cadre général de preuve qui englobe tous les résultats connus auparavant.

Notons que Robert (1976, 1987) a étudié le cas particulier des algorithmes asynchrones sans retards: $\tau_j^i(t) = t$, pour $t \in T^i$. Ce cas correspond à une

implantation parallèle asynchrone du processus de "conduite libre", vu à la section 2.2, où le choix des séquences de mises à jour se fait par les processeurs de manière dynamique.

Dans cette section, nous énonçons d'abord le théorème général de convergence tel que reformulé dans Bertsekas et Tsitsiklis (1989a, 1989b). Ensuite, nous l'appliquons au cas des fonctions contractantes et à celui des fonctions monotones. Nous décrivons aussi brièvement des exemples tirés de la recherche opérationnelle, qui s'inscrivent dans les deux cadres. Finalement, nous comparons la vitesse de convergence des algorithmes synchrones et asynchrones.

4.1 Théorème général de convergence

Ce théorème est dû à Bertsekas (1983) et a été reformulé différemment par Bertsekas et Tsitsiklis (1989a, 1989b). C'est cette dernière version que nous présentons.

Théorème 1: *Supposons que pour chaque* $i \in \{1, ..., p\}$, *il existe une suite* $\{X_i(k)\}$ *de sous-ensembles de* X_i *telle que:*

(a) $X_i(k+1) \subset X_i(k) \quad , \forall\, k \geq 0$

(b) *Les ensembles* $X(k) = \prod_{i=1}^{p} X_i(k)$ *sont tels que* $f(x) \in X(k+1)\,, \forall\, x \in X(k)$.

(c) *Tout point limite* x^* *d'une suite* $\{x(k)\}$ *avec* $x(k) \in X(k)$, *est un point fixe de* f.

De plus, si $x(0) \in X(0)$ et si l'hypothèse 2.1 (asynchronisme total) est vérifiée, alors tout point limite de la suite $\{x(t)\}$ générée par l'itération asynchrone (3.1)-(3.2) est un point fixe de f.

Les conditions suffisantes (a), (b) et (c) de ce théorème sont aussi nécessaires pour les cas suivants: f est linéaire (Chazan et Miranker, 1969), le domaine X de f est fini (Uresin et Dubois, 1990).

Le théorème est dit général, car il suffit de construire les ensembles $X_i(k)$ ayant les propriétés (a)-(c) pour conclure à la convergence. Notons que cette construction n'est pas toujours évidente. Nous allons illustrer ce théorème en l'appliquant au cas des fonctions contractantes et à celui des fonctions monotones.

4.2 Fonctions contractantes

Considérons la norme suivante sur $\Re^n$:

$$\| x \| = \max_{1 \le i \le p} \frac{\|x_i\|_i}{w_i} \qquad , x \in \Re^n$$

où $x = (x_1, ..., x_p)$;

$x_i \in \Re^n$;

$\| . \|_i$ est une norme sur $\Re^n$;

w_i est réel positif.

Supposons que f a la propriété de contraction suivante. Il existe $\alpha \in [0,1)$ tel que

$$\| f(x) - x^* \| \le \alpha \| x - x^* \| \qquad , \forall x \in \Re^n \tag{4.1}$$

où x^* est un point fixe de f.

Soit $x(0) \in X$ une solution initiale, définissons $X_i(k)$ par:

$$X_i(k) = \left\{ x_i \in \Re^{n_i} \mid \| x_i - x_i^* \|_i \leq \alpha^k \| x(0) - x^* \| \right\}$$

Les ensembles $X_i(k)$ vérifient les conditions du théorème et l'algorithme totalement asynchrone converge vers un point fixe de f.

4.3 Exemples de fonctions contractantes

Ces exemples sont discutés dans Bertsekas (1983) et Bertsekas et Tsitsiklis (1989a).

- Solution des systèmes d'équations linéaires. Dans ce cas, $f(x) = Ax + b$, où A est une matrice carrée de $\Re^n$, telle que le rayon spectral (la plus grande valeur propre) de $| A |$ (valeur absolue de A) est inférieur (strictement) à 1. Un exemple est le calcul d'un vecteur ligne $\pi = \pi P$, où P est la matrice de transition d'un processus de Markov et π représente les probabilités à l'état stationnaire. Il faut dans ce cas fixer une des composantes de π pour que la condition de contraction soit vérifiée.
- Méthode du gradient en optimisation sans contraintes. Soit $F : \Re^n \rightarrow \Re$, une fonction objectif deux fois continue et différentiable dont le hessien est borné et vérifie la condition de dominance de la diagonale. L'itération du gradient $x := x - \alpha \nabla F(x)$ converge de manière asynchrone, pourvu que le pas α soit suffisamment petit.

– Les méthodes itératives possédant la propriété de contraction, utilisées pour résoudre des problèmes d'inégalités variationnelles définies sur un produit cartésien. Un exemple est le gradient projeté (Bertsekas et Tsitsiklis, 1989a)

4.4 Fonctions monotones

Considérons une fonction f de X vers X, monotone, ayant un point fixe unique et supposons qu'il existe $\underline{x}$ et $\overline{x}$ tels que $\underline{x} \leq f(\underline{x}) \leq f(\overline{x}) \leq \overline{x}$.

Soit f^k la composition de f avec elle-même k fois (f^o est l'identité). Définissons les ensembles

$$X(k) = \left\{ x \in X \mid f^k(\underline{x}) \leq x \leq f^k(\overline{x}) \right\}$$

$$X = \prod_{i=1}^{p} X_i$$

$$X_i(k) = \left\{ x_i \in X_i \mid f_i^k(\underline{x}) \leq x_i \leq f_i^k(\overline{x}) \right\}$$

Les conditions suffisantes du théorème général de convergence étant vérifiées, l'algorithme totalement asynchrone correspondant à une fonction monotone (vérifiant les conditions ci-dessus) converge vers le point fixe de celle-ci.

4.5 Exemples de fonctions monotones

Les conditions de la section 4.3 sont vérifiées par des algorithmes de programmation dynamique tel que l'algorithme de Bellman-Ford pour la recherche des

chemins les plus courts (Bertsekas, 1982). Elles le sont aussi pour les méthodes de relaxation duale appliquées aux problèmes de flots dans les réseaux, à coûts linéaires et non-linéaires (Bertsekas et Tsitsiklis, 1989a; Bertsekas et Castanon, 1991; Bertsekas et El Baz, 1987; Chajakis et Zenios, 1991).

Considérons le problème du chemin le plus court dans un graphe orienté $G = (N, A)$, où N est l'ensemble des sommets et A l'ensemble des arcs. On veut trouver l'arborescence des chemins les plus courts reliant chaque sommet i au sommet 1. Supposons que tous les cycles sont positifs, la longueur des plus courts chemins est la solution unique des équations

$$x_i = \min_{\{j|(i,j)\in A\}} (a_{ij} + x_j) = f_i(x)$$

$$x_1 = 0$$

où a_{ij} est la longueur de l'arc (i, j).

L'algorithme de Bellman-Ford résoud itérativement l'équation $x_i := f_i(x)$. Sa version asynchrone converge (Bertsekas, 1982, 1983; Bertsekas et Tsitsiklis, 1989a). Bertsekas et Tsitsiklis (1989a) comparent les versions synchrones et asynchrones de cet algorithme. En supposant qu'une mise à jour prenne moins de temps que le délai de la communication entre deux processeurs et que ce dernier est constant, ils montrent que la version asynchrone converge toujours avant la version synchrone.

Lorsque les délais de communication sont arbitraires, Bertsekas et Tsitsiklis (1989a, p. 450) présentent un exemple (dû à Gafni, M. et Gallager, R. G.), où l'algorithme totalement asynchrone de Bellman-Ford peut produire un nombre exponentiel d'échanges de messages, alors que la version synchrone ne nécessite pas plus de n^3 échanges. Toutefois, les auteurs précisent qu'il ne s'agit là que d'un pire cas exceptionnel et que le comportement expérimental en moyenne est linéaire en n.

4.6 Comparaison des vitesses de convergence des algorithmes synchrones et asynchrones

Bertsekas et Tsitsiklis (1989b) comparent les itérations synchrones et asynchrones dans les cas des fonctions monotones et des fonctions contractantes. Ils démontrent que les algorithmes asynchrones convergent au moins aussi vite que leurs homologues synchrones. Dans le cas des fonctions contractantes, l'avantage des algorithmes asynchrones croît lorsque le nombre de variables nécessaires à une mise à jour décroît.

5. Algorithmes partiellement asynchrones

5.1 Exemples

Les algorithmes totalement asynchrones, étudiés à la section 4, n'imposent aucune borne sur les délais des communications. Ces derniers peuvent tendre à l'infini et compromettre la convergence de certains algorithmes. La convergence sera pourtant assurée juste par le fait d'imposer une borne B, finie mais arbitraire, sur les délais de communication. Nous illustrons cette situation par l'exemple suivant (Bertsekas et Tsitsiklis, 1989a).

Exemple 1. Nous voulons résoudre $x = Ax$ où $A = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}$.

L'itération synchrone $x(t+1) := Ax(t)$ converge en une itération. L'algorithme totalement asynchrone ne converge pas car A a une valeur propre égale à 1. Nous donnons une autre preuve qui ressort l'effet des délais de communications. Considérons le scénario suivant. Deux processeurs effectuent chacun la mise à jour d'une composante. Aux instants $t_1, t_2, ...$, chaque processeur envoie la valeur de sa composante que l'autre reçoit avec un délai nul et l'utilise immédiatement dans les calculs.

On a alors:

$$x_1(t+1) = \frac{x_1(t)}{2} + \frac{x_2(t_k)}{2} \qquad , t_k \leq t < t_{k+1}$$

$$x_2(t+1) = \frac{x_1(t_k)}{2} + \frac{x_2(t)}{2} \qquad , t_k \leq t < t_{k+1}$$

Ainsi,

$$x_1(t_{k+1}) = \left(\frac{1}{2}\right)^{t_{k+1}-t_k} x_1(t_k) + \left(1 - \left(\frac{1}{2}\right)^{t_{k+1}-t_k}\right) x_2(t_k)$$

$$x_2(t_{k+1}) = \left(\frac{1}{2}\right)^{t_{k+1}-t_k} x_2(t_k) + \left(1 - \left(\frac{1}{2}\right)^{t_{k+1}-t_k}\right) x_1(t_k)$$

En faisant la soustraction,

$$| x_2(t_{k+1}) - x_1(t_{k+1}) | = (1 - \varepsilon_k) | x_2(t_k) - x_1(t_k) |$$

où $\varepsilon_k = 2\left(\frac{1}{2}\right)^{t_{k+1}-t_k}$. Même si $| x_2(t_k) - x_1(t_k) |$ décroît toujours, $x(t)$ ne converge pas, sauf si $\prod_{k=1}^{\infty} (1 - \varepsilon_k) = 0$. En effet, $\prod_{k=1}^{\infty} \left(1 - k^{-2}\right) > 0$ et si on choisit $t_{k+1} - t_k$ tel que $\varepsilon_k < k^{-2}$, l'algorithme totalement asynchrone ne converge pas (voir figure 1).

Toutefois, si l'hypothèse 3.2 (asynchronisme partiel) est valide, alors $t_{k+1} - t_k \leq B$ et

$$\prod_{k=1}^{\infty} (1 - \varepsilon_k) = 0.$$

Cet exemple est typique des algorithmes qui convergent sous l'hypothèse d'asynchronisme partiel (pour tout B fini) et qui divergent en cas d'asynchronisme total. Dans de tels algorithmes, il y a souvent une fonction-distance qui mesure l'éloignement entre la solution courante et l'ensemble des points fixes. Durant l'évolution de ces algorithmes, cette distance se contracte d'un facteur inférieur à

1 et qui dépend de B. Si B tend vers l'infini, ce rapport de réduction tend vers 1 et on ne se rapproche plus de l'ensemble des points fixes. Ainsi, la convergence n'est plus garantie.

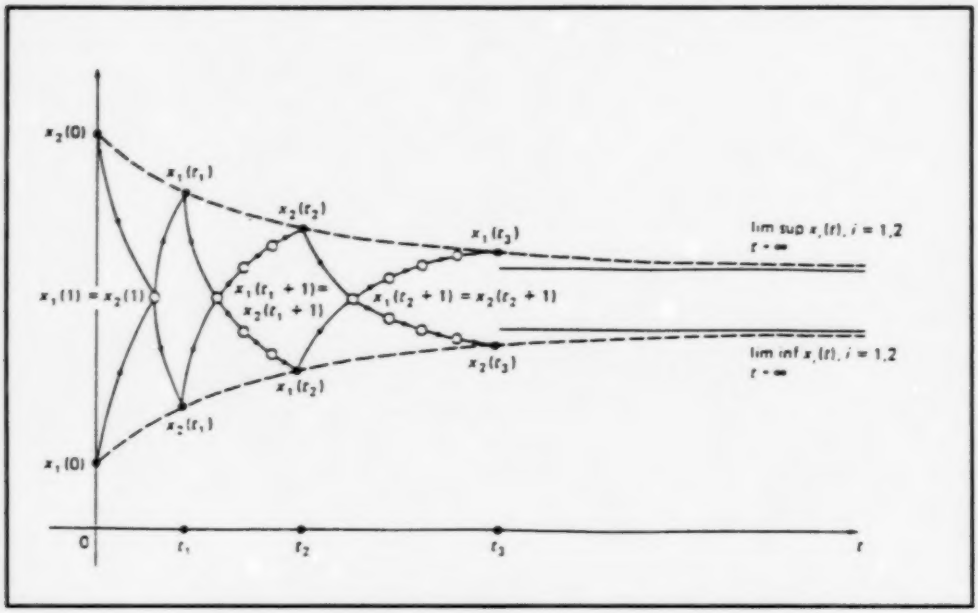


Figure 1 Illustration de la divergence obtenue avec l'exemple 1

La convergence des algorithmes partiellement asynchrones a été obtenue pour divers problèmes, par plusieurs auteurs: Bertsekas et Tsitsiklis (1989a), pour les problèmes "d'entente" (agreement) et de balancement de la charge des processeurs; Lubachevsky et Mitra (1986), pour le problème du point fixe d'une matrice non négative à rayon spectral unitaire, dont un exemple est la recherche du vecteur des probabilités π à l'état stationnaire d'un processus de Markov avec une matrice de transition P ($\pi := \pi P$). Tseng, Bertsekas et Tsitsiklis (1990) ont

étudié le cas où la fonction définissant le processus itératif est non-expansive avec application à la méthode de relaxation duale, pour résoudre les problèmes de flots à coût strictement convexe.

5.2 Un résultat de convergence

Considérons des itérations asynchrones sous l'hypothèse d'asynchronisme partiel. $x(t+1)$ dépend uniquement de $x(t), x(t-1), ..., x(t-B+1)$. L'information pertinente à l'instant t est décrite par $z(t) \in Z = X^B$, $z(t) = (x(t), x(t-1), ..., x(t-B+1))$, où X^B dénote le produit cartésien de X, B fois.

Soit X^* l'ensemble des points fixes de la fonction f définissant le processus itératif. Z^* est l'ensemble des éléments de Z définis par $(x^*, ..., x^*)$ où $x^* \in X^*$.

Le théorème suivant est dû à Bertsekas et Tsitsiklis (1989a):

Théorème 2: *Considérons les itérations asynchrones (3.1)-(3.2) et supposons que f est continue et que l'hypothèse 3.2 d'asynchronisme partiel est vraie. Supposons aussi l'existence d'un instant de mise à jour t^* et d'une fonction $d : Z \to [0, \infty)$ tels que:*

(a) *Pour tout $z(0) \notin Z^*$, on a $d(z(t^*)) < d(z(0))$.*

(b) *Pour tout $z(0) \in Z$, pour tout $t \geq 0$, on a $d(z(t+1)) \leq d(z(t))$.*

Alors, tout point limite z^* de la suite $\{z(t)\}$ est élément de Z^*.

La fonction $d : Z \to [0, \infty)$, définie dans le théorème précédent, mesure l'éloignement de l'ensemble des points fixes Z^*. L'instant t^* garantit que cette

distance décroît après au plus t^* mises à jours. Nous donnons un exemple dans la sous-section suivante.

5.3 Application au problème d'entente

Il s'agit de résoudre de manière asynchrone le problème suivant: $x = A\,x$ où les coefficients a_{ij} de la matrice non négative $A\,(n \times n)$ sont tels que $\sum_{j=1}^{n} a_{ij} = 1$, pour tout i.

Les processeurs vont s'échanger les valeurs de x et un processeur i change x_i en effectuant une combinaison linéaire convexe de la valeur de x qu'il possède. La convergence est atteinte lorsque les processeurs ont la même valeur du vecteur x. On dit que les processeurs se sont mis d'accord sur une même valeur de x (entente ou "agreement").

La fonction d définie par:

$$d(z(t)) = \max_i \max_{t-B<\tau\leq t} x_i(\tau) - \min_i \min_{t-B<\tau\leq t} x_i(\tau)$$

vérifie les propriétés du théorème de la sous-section 5.3 pourvu qu'une des entrées diagonales de A soit non-nulle (Bertsekas et Tsitsiklis, 1989a). Les auteurs montrent aussi que la convergence est réalisée après au plus $2nB$ mises à jour. La convergence est donc linéaire en n.

6. Algorithmes faiblement asynchrones

Nous nous intéressons maintenant aux algorithmes partiellement asynchrones qui ne convergent que lorsque la mesure d'asynchronisme B (voir hypothèse

3.1) est faible. Des exemples de cette classe sont les méthodes de descente en optimisation sans contraintes. À notre connaissance, Bertsekas et Tsitsiklis (1989a) sont les seuls à avoir étudié la convergence de ces algorithmes. Rappelons que les méthodes de type gradient, convergent de façon totalement asynchrone seulement si le pas est suffisamment petit et que le hessien de la fonction à minimiser a une diagonale dominante (voir la section 3).

Nous illustrons le comportement des algorithmes qui ne tolèrent qu'un asynchronisme faible, à l'aide de l'exemple suivant (Bertsekas et Tsitsiklis, 1989a).

Exemple 2. Considérons l'itération $x := x - \alpha A x$ où α est un pas positif et

$$A = \begin{bmatrix} 1+\epsilon & 1 & 1 \\ 1 & 1+\epsilon & 1 \\ 1 & 1 & 1+\epsilon \end{bmatrix}, \quad 0 < \epsilon < 1.$$

Cette itération est la même que $x := x - \alpha \nabla F(x)$, où $F(x) = \frac{1}{2} x^T A x$. La version synchrone converge pourvu que α soit suffisamment petit. La version totalement asynchrone ne converge pas, car le rayon spectral de la matrice $|I - \alpha A|$ est toujours supérieur à 1, pour tout α positif et ϵ entre 0 et 1. Considérons le scénario suivant d'un algorithme partiellement asynchrone. Trois processeurs effectuent des mises à jour, une composante est affectée à chacun. Aux instants t_1 , t_2 , ... , avec $t_k = kB$, chaque processeur envoie la valeur de sa composante que les autres reçoivent avec un délai nul et utilisent dans les calculs. On a alors

$$x_1(t+1) = x_1(t) - \alpha((1+\epsilon)x_1(t) + x_2(t_k) + x_3(t_k)), \quad t_k \leq t < t_{k+1} \quad (6.1)$$

x_2 et x_3 vérifient des équations similaires. Nous donnons ici une analyse de convergence facile à généraliser, mais qui a toutefois un aspect qualitatif (une analyse de cet exemple à l'aide des valeurs propres se trouve dans Bertsekas et Tsitsiklis (1989a, p. 487)). Pour $t \in [t_k, t_{k+1})$, les différences $x_i(t) - x_i(t_k)$ sont de l'ordre de αB (car à chaque mise à jour, on bouge de α). Donc, la différence entre la mise à jour (6.1) et celle synchrone

$$x_1(t+1) = x_1(t) - \alpha((1+\epsilon)x_1(t) + x_2(t) + x_3(t))$$

est de l'ordre de $\alpha^2 B$. Ainsi, tant que αB est petit devant 1, l'effet de l'asynchronisme sera faible devant celui des mises à jour et l'itération partiellement asynchrone peut être approximée par l'algorithme synchrone (car $\alpha^2 B < \alpha$). Alors que si αB est grand devant 1, l'itération partiellement asynchrone peut s'éloigner sensiblement de l'itération synchrone, provoquant ainsi la divergence. Donc, lorsque le degré d'asynchronisme est faible, l'algorithme partiellement asynchrone converge.

L'analyse que nous avons entrepris dans l'exemple 2, se généralise à la classe des méthodes de descente et conduit à des algorithmes faiblement asynchrones (Bertsekas et Tsitsiklis, 1989a).

7. Terminaison des algorithmes asynchrones

Nous considérons toujours la classe des algorithmes itératifs. Ces derniers doivent s'arrêter dans un temps fini lorsqu'une conditon est safisfaite. Dans le

cas synchrone, ce problème est facile à facile à résoudre pourvu que les délais de communication soient bornés. Par contre, il est beaucoup plus difficile dans le cas des algorithmes asynchrones. Ceci est dû au fait que les processeurs n'ont qu'une connaissance partielle du progrès réalisé par l'algorithme au niveau global. Le problème peut être décomposé en deux niveaux: la détection de la terminaison et sa réalisation.

7.1. Détection de la terminaison

Des algorithmes de détection de la terminaison existent (Dijkstra et Scholten, 1980; Chandy et Lamport, 1985). Ils sont décrits dans Bertsekas et Tsitsiklis (1989a). Ces méthodes de détection imposent toutefois des restrictions. L'état de terminaison doit avoir les deux propriétés suivantes:

1. aucun message n'est en transit sur des liens de communications;
2. une mise à jour par un processeur i ne change pas la valeur de x_i.

7.2. Réalisation de la terminaison

Même si un algorithme asynchrone termine en un nombre fini d'itérations (par exemple l'algorithme de Bellman-Ford pour les plus courts chemins), il n'est pas nécessairement vrai que cette terminaison vérifie les propriétés utiles à sa détection (voir 7.1.). Pour que ce soit le cas, les hypothèses suivantes sont introduites (Bertsekas et Tsitsiklis, 1989b):

1. les communications se font par échanges de messages;

2. si un processeur i a envoyé $x_i(t)$ à un processeur j, alors il ne lui envoie un nouveau message que si x_i change;
3. les messages sont reçus dans l'ordre de leur envoi;
4. chaque processeur envoie au moins un message aux autres.

Considérons maintenant les algorithmes asynchrones qui ne terminent jamais s'il n'y a pas de condition d'arrêt. Dans le cas synchrone, le processus itératif $x(t+1) := f(x(t))$ est arrêté lorsque $\|x(t+1) - x(t)\| \leq \epsilon$, où ϵ est une tolérance donnée. Une idée naturelle est de généraliser ce critère d'arrêt au cas asynchrone en utilisant $g(x)$ à la place de $f(x)$ dans le processus itératif $x := f(x)$, où $g : X \to X$ est définie par :

$$g_i(x) = \begin{cases} f_i(x) & si \ \|f_i(x) - x_i\| \geq \epsilon \\ x_i & sinon. \end{cases}$$

Malheureusement, même si $x := f(x)$ converge de manière asynchrone, l'algorithme asynchrone correspondant à $x := g(x)$ peut itérer indéfiniment. Nous donnons l'exemple suivant comme preuve.

Exemple 3. $f : \Re^2 \to \Re^2$ est définie par

$$\begin{aligned} f_1(x) &= \begin{cases} -x_1 & si\, x_2 \geq \frac{\epsilon}{2} \\ 0 & si\, x_2 < \frac{\epsilon}{2} \end{cases} \\ f_2(x) &= \frac{x_2}{2} \end{aligned}$$

L'itération asynchrone $x := f(x)$ convege vers $(0,0)$. Considérons maintenant l'itération $x := g(x)$. Si on part de x_2 entre $\frac{\epsilon}{2}$ et ϵ, x_2 ne changera jamais et le processeur 1 effectuera indéfiniment des mises à jour $x_1 := -x_1$ (on suppose qu'initialement $|x_1| > \frac{\epsilon}{2}$).

Bertsekas et Tsitsiklis (1989b) donnent des conditions suffisantes pour que l'itération $x := g(x)$ termine en temps fini. Soit $I \subset \{1,\ldots,p\}$ et $\theta \in X$. Considérons l'itération $x := f^{I,\theta}(x)$ où

$$f_i^{I,\theta}(x) = \begin{cases} \theta_i & si \quad i \in I \\ f_i(x) & sinon \end{cases}$$

Proposition: *Si pour tout I et θ, l'itération $x := f^{I,\theta}(x)$ converge de manière asynchrone, alors l'itération asynchrone $x := g(x)$ termine en temps fini.*

Les hypothèses de cette proposition sont vérifiées si f est fortement contractante:

$$\|f(x) - f(y)\| \leq \alpha\|x - y\| \,, \; \forall \; x, y \in \Re^n$$

avec $\alpha \in [0,1)$. L'itération asynchrone $x := g(x)$ terminera alors en temps fini.

8. Conclusion

Nous avons présenté une synthèse de la littérature qui traite des algorithmes parallèles asynchrones en recherche opérationnelle. La plupart des recherches faites dans ce domaine concernent les méthodes itératives de la forme $x := f(x)$. Nous avons d'abord analysé des implantations parallèles synchrones de ces méthodes. Nous avons vu que ces implantations présentent des inconvénients dûs à la synchronisation. Pour pallier à ces défauts, nous avons considéré des implantations asynchrones. Ces dernières sont divisées en trois classes: totalement, partiellement et faiblement asynchrones. Cette distinction est introduite pour assurer la convergence de certaines méthodes, convergence qui peut dépendre du

degré d'asynchronisme permis. Nous avons présenté, pour chaque classe, une synthèse des principaux résultats de convergence et nous les avons illustrés par des méthodes connues en recherche opérationnelle. Outre la convergence, nous avons discuté le problème de la terminaison des algorithmes asynchrones.

L'analyse théorique montre que l'approche asynchrone est adéquate pour les machines parallèles asynchrones où la communication est coûteuse. C'est le cas des systèmes multiprocesseurs et des réseaux distribués tels les réseaux de Transputers et les réseaux de mini-ordinateurs. Remarquons que ce genre de machines, d'un coût relativement peu élevé, va tirer davantage du calcul asynchrone que des systèmes SIMD beaucoup plus coûteux comme la Connection Machine.

Nous avons remarqué une marge assez importante entre les études théoriques et pratiques. Peu de travaux empiriques ont été réalisés (Bertsekas et Castanon, 1991; Chajakis et Zenios, 1991), alors que beaucoup de méthodes ont été abordées uniquement sur le plan théorique. Des tests numériques devraient être entrepris afin de comprendre et de mesurer l'impact des architectures des machines parallèles sur le comportement des méthodes asynchrones.

Outre l'aspect empirique, plusieurs avenues de recherche théorique restent à explorer. D'abord, les études de convergence se sont limitées aux méthodes itératives de point fixe, et ce serait très intéressant de les généraliser. Même dans

ce cadre, on peut penser à assouplir les conditions de convergence. Enfin, la terminaison des algorithmes asynchrones constitue un problème difficile où peu de recherche est faite.

Références

Awerbuch, B. (1985), "Complexity of network synchronization", *J. ACM*, 32, 804–823.

Baudet, G. M. (1978), "Asynchronous iterative methods for multiprocessors", *J. ACM*, 2, 226–244.

Bertsekas,D. P. (1982), "Distributed dynamic programming", *IEEE Trans. Aut. Control.*, AC-27, 610–616.

Bertsekas, D. P. (1983), "Distributed asynchronous computation of fixed points", *Math. Programm.*, 27, 107–120.

Bertsekas, D. P. and Castanon, D. A. (1991), "Parallel synchronous and asynchronous implementations of the auction algorithm", *Parallel Computing*, 17, 707–732.

Bertsekas, D. P. and El Baz, D. A. (1987), "Distributed asynchronous relaxation methods for convex network flow problems", *SIAM J. Control Optimization*, 25, 74–84.

Bertsekas, D. P. and Tsitsiklis, J. N. (1989a), "Parallel and Distributed Computation: Numerical Methods", *Prentice-Hall, Englewood Cliffs, NJ.*

Bertsekas, D. P. and Tsitsiklis, J. N. (1989b), "Convergence rate and termination of asynchronous iterative algorithms", *Proc. 1989 Int. Conf. on Supercomputing*, Greece,1989, 461–470.

Chabini, I. (1990) , "Des implantations parallèles de l'algorithme d'approximation linéaire pour la résolution du problème d'affectation du trafic", Mémoire de maîtrise, Département d'informatique et de recherche opérationnelle, Université de Montréal, Canada.

Chajakis, E. D., and Zenios, S. A. (1991) , "Synchronous and asynchronous implementations of the relaxation algorithms for nonlinear network optimization",

Parallel Computing, 17, 873–894.

Chandy, K. M. and Lamport,L. (1985) "Distributed snapshots:determining global states of distributed systems",*ACM Trans. Comput. Syst.*, 3, 63–75.

Chazan, D. and Miranker, W. (1969) "Chaotic relaxation",*Linear Algebra and its Applications*, 2, 199–222.

Dijkstra, E. W. and Scholten, C. S. (1980) "Termination detection for diffusing computation", *Inform. Process. Lett.*, 11, 1–4.

Donnelly, J. D. P. (1971), "Periodic chaotic relaxation", *Linear Algebra and its Applications*, 4, 199–222.

Dubois, M. and Briggs, F. A. (1982) "Performence of synchronized iterative processes in multiprocessor systems", *IEEE Trans. Software Eng.*, 8, 419–431.

El Tarazi, M. N. (1982) "Some convergence results for asynchronous algorithms", *Numerische Mathematik*, 39, 325–340.

Kung, H. T. (1976) "Synchronized and asynchronous parallel algorithms for multiprocessors", *Algorithms and complexity*, Academic Press, New York, 153–200.

Lint, B. and Agerwala, T. (1981) "Communication issues in the design and analysis of parallel algorithms", *IEEE Trans. Software Eng.*, SE-7, 174–188.

Lubachevsky, B. and Mitra, D. (1986) "A Chaotic asynchronous algorithm for computing the fixed point of a nonnegatif matrix of unit spectral radius", *J. ACM*, 33, 130–150.

Miellou, J. C. (1975a) "Itérations chaotiques à retards, études de la convergence dans le cas d'espaces partiellement ordonnés", *Comptes Rendus, Academie de Sciences de Paris*, 280, Série A, 233–236.

Miellou, J. C. (1975b) "Algorithmes de relaxation chaotique à retards", *R.A.I.R.O.*, 9, R-1, 55–82.

Miellou, J. C. and Spiteri, P. (1985) "Un critère de convergence pour des méthodes générales de point fixe", *Math. Modelling Numer. Anal.*, 19, 645–669.

Robert, F. (1976) "Contraction en norme vectorielle: convergence d'itérations chaotiques pour des équations non linéaires de point fixe à plusieurs variables", *Linear Algebra and its applications*, 13, 19–35.

Robert, F. (1987) "Itérations discrètes asynchrones", Rapport technique 671M, I.M.A.G., Université de Grenoble, France.

Smart, D. and White J. (1988) "Reducing the parallel solution of sparse circuit matrices using reordred Gaussian elimination and relaxation", *Proc. 1988 ISCAS*, Espoo, Finland.

Tseng, P., Bertsekas, D. P., and Tsitsiklis, J. N. (1990) "Partially asynchronous, parallel algorithms for the network flow and other problems", *SIAM J. Control and Optimization*, 28, 678–710.

Tsitsiklis, J. N. (1989) "A comparaison of Jacobi and Gauss-Seidel parallel iterations", *Applied Math. Lett.*, 2,167–170.

Uresin, A. and Dubois M. (1986) "Generalized asynchronous iterations", *Lecture Notes in Computer Science*, 237, Springer, Berlin, 272–278.

Uresin, A. and Dubois M. (1990) "Parallel asynchronous algorithms for discrete data", *J. ACM*, 37, Springer, 588–606.